1. Overview of the system:

The program that will be explained is an interactive gesture-controlled application called Fruit Orchard. This system is designed as a game and educational tool to demonstrate how hand gestures can interact with a graphical user interface (GUI). By using their hand users can interact with the system and play the Fruit Orchard game. The purpose of the game is picking the fruits by hand and moving it towards the basket and making the highest score. During the game, users will try to find apples by scrolling the screen to the right, left, up and down using predefined hand movements, and then they will try to select these apples with their hands and scroll apples to the basket to increase their score. To demonstrate more, check the descriptions below that explains which gestures are chosen, by which landmarks, and for which interactions.

1.1 Gesture Recognition:

By using Mediapipe's hand solutions the program detects hand gestures and give appropriate output. There are 3 different key gestures in the system which triggers different functionalities.

a) One finger gesture (Index): One-finger gesture allows users to scroll the screen left, right, up and down. Thanks to this feature, users can easily find apples on the page or move in different directions on the pages. Basically, it activates scroll mode for navigating the canvas.

b) Two finger gesture (Index + Middle): Two-finger gesture allows users to press buttons such as play game and instrucitons and at the same time, users can select apples and drag them to the basket within the game with using two finger gesture. Basically, it controls the virtual cursor for hovering over items.

c) Three finger gesture (Index + Middle + Ring): The application can be closed directly using the three-finger gesture. It executes a quit command which directly closes the application.

Gesture	Landmark	Functionality
One Finger	Index Finger	Scroll Mode
Two Finget	Index + Middle Fingers	Cursor Control Mode
Three Finger	Index+ Middle + Ring Fingers	Quit

Figure 1 : Gesture Functionality Table

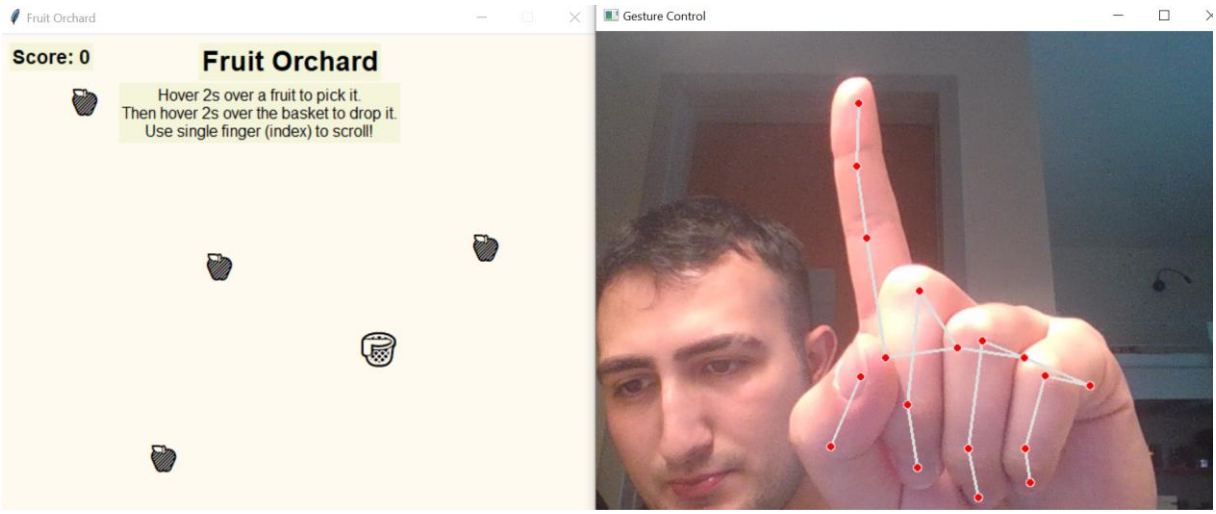

Figure 2: One Finger Gesture

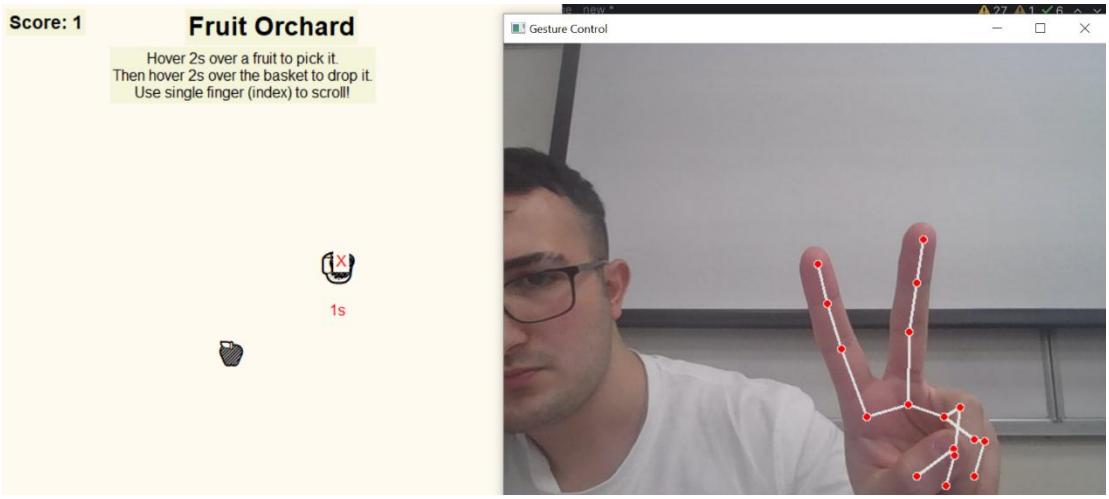

Figure 3: Two Finger Gesture

1.2 Landmarks:

The program utilizes Mediapipe Hand Landmarks especially it focuses on the following for each finger:
Thumb: THUMB_TIP, THUMB_IP
Index Finger: INDEX_FINGER_TIP, INDEX_FINGER_PIP
Middle Finger: MIDDLE_FINGER_TIP, MIDDLE_FINGER_PIP
Ring Finger: RING_FINGER_TIP, RING_FINGER_PIP
Pinky Finger: PINKY_TIP, PINKY_PIP

The landmark positions are analyzed to determine whether a finger is extended:
Thumb: Compared using x-coordinates (THUMB_TIP.x > THUMB_IP.x).
Other Fingers: Compared using y-coordinates (FINGER_TIP.y < FINGER_PIP.y).

1.3 Interaction It Triggers

As explained in the gesture recognition part, one finger trigers scroll interaction it enables users to scrolls the canvas horizontally or vertically based on the relative movement of the hand.Two Finger trigers cursor interaction it enables users to move a virtual cursor to hover over interactive elements (e.g. fruits or menu buttons) and also it activates a selection timer for hovered items which allows users to pick fruits or navigate menus. Three Finger triggers quit interaction. By using three finger users close the application by initiating the on_close() function.

1.4 GUI

A user-friendly interface has been developed for the Fruit Orchard which allows users to navigate easily between different pages by using hand interaction and also interface provide information on how to play the game for an inexperienced user. The Fruit Orchard games GUI effectively follows human-computer interaction principles. It provides intuitive gesture-based controls and clear visual feedback mechanisms.

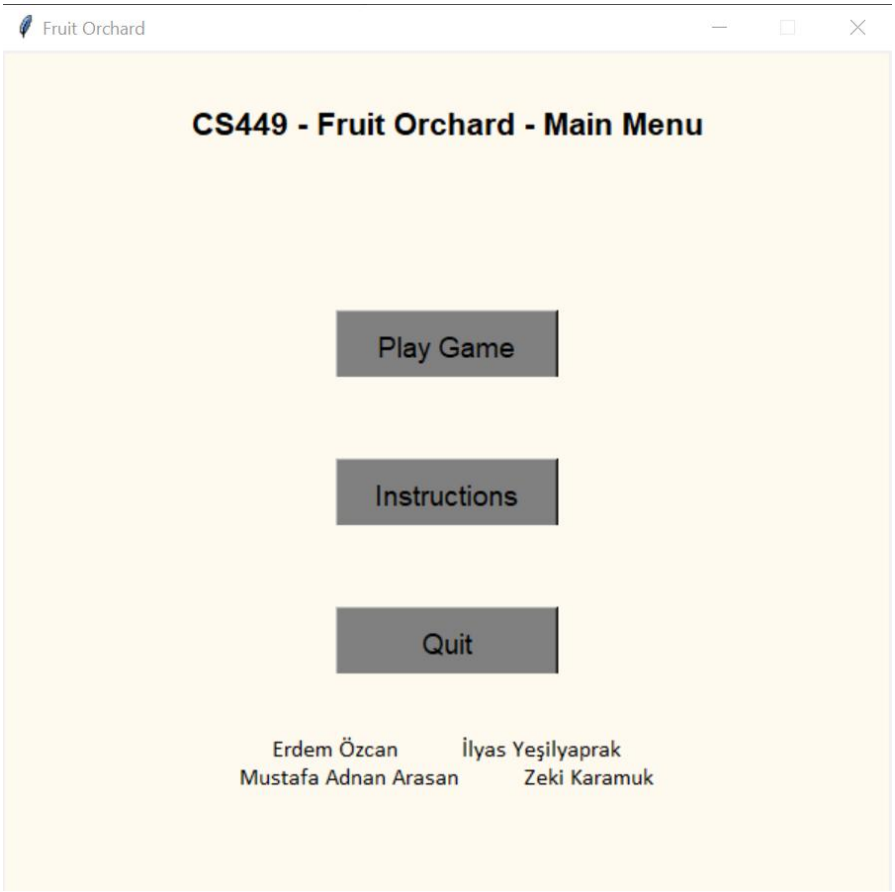

Figure 4: Main Menu Screen

In figure 5, the instrucitons screen is shown. Instruction screen explains how to interact with the system and locate the fingers in a right position which aligns with Norman's emphasis on clear signifiers to guide user actions. And also when the user hovers on the button, the buttons color is changed into blue and system shows the time needed to interact. By this way system gives informative feedback which ensures users to understand their interactions in real time and by this way the error that can be occur during the interaction is minimized which aligns with Shneiderman principles. Also the buttons are clearly labeled for navigation ("Play Game," "Instructions," "Quit"). This design follows Landauer's (1995) focus on usability and

usefulness which minimizes cognitive load. Overall, the GUI aligns with foundational principles in HCI and ensures both functionality and user satisfaction.

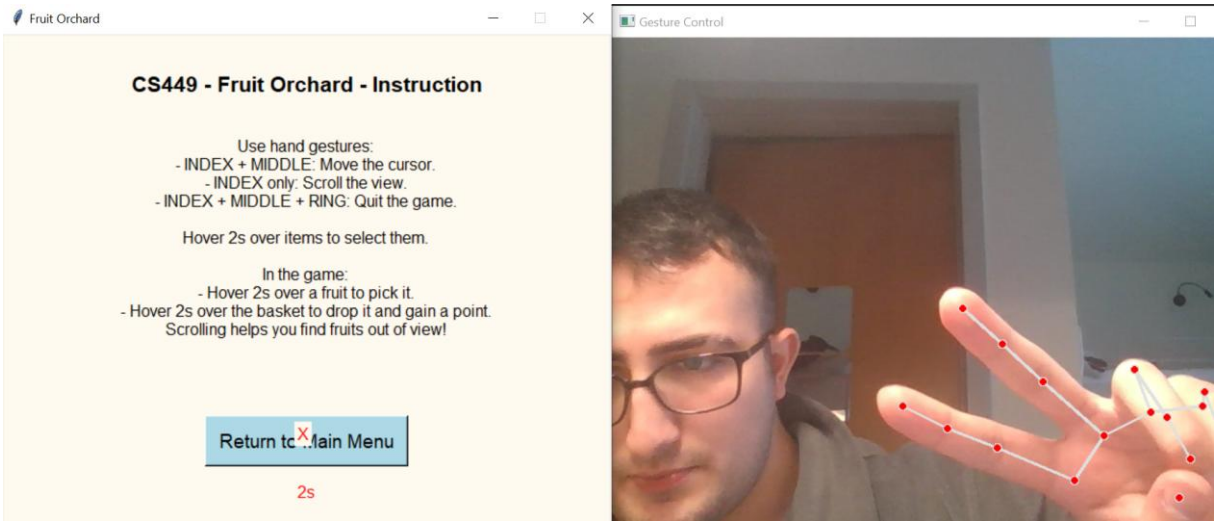

Figure 5: Instructions Page

2. Implementation Details:

The code implements a circular virtual keyboard system using gesture recognition through MediaPipe and OpenCV.

2.1 Landmark Detection and Processing

Webcam Input:

- The system captures frames from a webcam using OpenCV's `cv2.VideoCapture` method. The video feed resolution is set to a clear and consistent size (e.g., `1280x720`) to ensure that hand landmarks are accurately detected.

- Each frame is flipped horizontally to provide a mirror-like experience. This design choice aligns the visual feedback with the user's natural movements, creating an intuitive interaction system.

```python
def process_video():  1 usage
    """
    Captures video frames from the webcam, detects hand landmarks,
    and interprets gestures. Calls UI updates accordingly.
    """
    global hand_present, scroll_mode, prev_cursor_x, prev_cursor_y, running
    cap = cv2.VideoCapture(0)
    with mp_hands.Hands(
        static_image_mode=False,
        max_num_hands=1,
        min_detection_confidence=0.7,
        min_tracking_confidence=0.7,
    ) as hands:
        while running and cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            frame = cv2.flip(frame, 1)
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = hands.process(rgb_frame)

            cursor_visible = False
```

Figure 6 : Webcam Implementation

MediaPipe Hand Module:

- The code uses the MediaPipe Hands solution, which identifies 21 hand landmarks. These landmarks represent specific points such as fingertips, joints, and the wrist.

Normalized Coordinates:

- MediaPipe provides landmark coordinates normalized to the range `[0, 1]`, where:

  - `x` and `y` represent the horizontal and vertical positions relative to the frame dimensions.

  - `z` represents the depth (distance from the camera), though this may not be used in gesture recognition if not coded.

```python
def is_finger_extended(hand_landmarks, finger_name):  4 usages
    """
    Checks if a particular finger is extended.
    For thumb, we compare x-coordinates (tip > pip means extended).
    For other fingers, we compare y-coordinates (tip < pip means extended).
    """
    mp_hands_tips = {
        'thumb': mp_hands.HandLandmark.THUMB_TIP,
        'index': mp_hands.HandLandmark.INDEX_FINGER_TIP,
        'middle': mp_hands.HandLandmark.MIDDLE_FINGER_TIP,
        'ring': mp_hands.HandLandmark.RING_FINGER_TIP,
        'pinky': mp_hands.HandLandmark.PINKY_TIP,
    }
    mp_hands_pips = {
        'thumb': mp_hands.HandLandmark.THUMB_IP,
        'index': mp_hands.HandLandmark.INDEX_FINGER_PIP,
        'middle': mp_hands.HandLandmark.MIDDLE_FINGER_PIP,
        'ring': mp_hands.HandLandmark.RING_FINGER_PIP,
        'pinky': mp_hands.HandLandmark.PINKY_PIP,
    }

    tip = hand_landmarks.landmark[mp_hands_tips[finger_name]]
    pip = hand_landmarks.landmark[mp_hands_pips[finger_name]]
    if finger_name == 'thumb':
```

Figure 7 : Landmarks Implementation

Mapping Landmarks to Screen:

- Normalized coordinates are scaled to the screen's width and height to map hand positions accurately to cursor movements or interaction points. This ensures that gestures correspond directly to screen interactions.

2.2 Gesture Recognition

The code implements several gestures by analyzing the relative positions of hand landmarks. Each gesture is defined based on specific conditions, such as the distance between landmarks or the relative orientation of fingers.

2.2.1 Move Gesture

- Definition: Recognized when only the index finger is extended.

- Implementation:

- The system identifies the index finger's tip (`landmark[8]`) and maps its position to the screen cursor.

- Other fingers are checked to ensure they are retracted (landmarks for fingertips are closer to the palm landmarks).

```
def move_cursor(screen_x, screen_y):  1 usage
    """
    Moves the on-screen cursor to (screen_x, screen_y).
    If a fruit is being dragged, move it along with the cursor.
    """
    if not running or canvas is None:
        return
    try:
        canvas.coords(cursor_window, screen_x, screen_y)
        if dragging_fruit is not None:
            canvas.coords(dragging_fruit['obj'], screen_x, screen_y)
    except tk.TclError:
        pass
```

Figure 8 : Move Cursor Implementation

2.2.2 Click Gesture

- Definition: A pinching motion, detected when the thumb (`landmark[4]`) and index fingertip (`landmark[8]`) are close together.

- Implementation:

- The Euclidean distance between the thumb and index fingertip is calculated.

- If the distance is less than a predefined threshold (e.g., `0.04`), a "click" action is registered.

2.2.3 Scroll Gesture

- Definition: Identified when the index and middle fingers are extended, while other fingers remain retracted.

- Implementation:

- The system tracks the vertical movement of the hand and uses it to scroll between pages of the keyboard.

```
    index_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
    middle_tip = hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP]

    cx = (index_tip.x + middle_tip.x) / 2
    cy = (index_tip.y + middle_tip.y) / 2
```

Figure 9 : Gesture Implementations

2.2.4 Stop Sign Gesture

- Definition: Detected when all fingers are extended, representing an open palm.

- Implementation:

- Each fingertip landmark is compared with its respective palm landmark. If all fingertips are positioned above their respective base points, the gesture is identified as a stop sign.

- Used for toggling features like Caps Lock.

2.3. Circular Virtual Keyboard Design:

Circular Layout

- Buttons are distributed evenly in a circular pattern around a central point. The system uses trigonometric calculations to position each button based on its angle and distance from the center.

Button Rendering

- The buttons are rendered dynamically in each frame. Text labels or symbols are assigned to each button to represent keyboard characters or commands.

Interaction with Buttons

- The system continuously checks if the cursor (controlled by hand movements) is hovering over a button.

- When a gesture like "Click" is detected, the system identifies the active button and performs the corresponding action (e.g., typing a character or triggering a function)

2.4. Real-Time Interaction

Main Loop

- The main loop processes each frame as follows:

1. Captures a frame from the webcam.
2. Detects hand landmarks using MediaPipe.
3. Recognizes gestures based on landmark positions.
4. Updates the GUI (e.g., cursor movement, button highlighting).
5. Renders the updated frame to the screen.

Feedback Mechanisms

- The system provides real-time feedback by highlighting buttons under the cursor and displaying the currently active gesture on the screen.

2.5. Error Handling and Stability

- Try-except blocks are used to handle potential errors during:

  - Webcam capture.

  - Landmark detection.

  - Gesture recognition.

  - GUI rendering.

- This ensures the application remains stable even under challenging conditions, such as poor lighting or rapid hand movements.

```python
def scroll_canvas(direction):  4 usages
    """
    Scrolls the canvas in the given direction (up/down/left/right)
    A minimum interval is required between scrolls to prevent too
    """
    global last_scroll_time
    if not running or canvas is None:
        return
    current_time = time.time()
    if current_time - last_scroll_time < 0.05:
        return
    last_scroll_time = current_time
    try:
        if direction == 'up':
            canvas.yview_scroll(-1, 'units')
        elif direction == 'down':
            canvas.yview_scroll(1, 'units')
        elif direction == 'left':
            canvas.xview_scroll(-1, 'units')
        elif direction == 'right':
            canvas.xview_scroll(1, 'units')
    except tk.TclError:
        pass
```

Figure 10 : Scroll Implementation

2.6. Optimization

- To maintain performance, the system limits hand detection to a single hand.

- Processing is streamlined to ensure smooth operation, even on devices with limited computational power.

3. Code

Github Link: https://github.com/arasanadnan/cs449-assignment5

4. Github

To store the project code, track contributions, manage version control and effectively develop with teammates GitHub is used throughout the project. The repository which is a centralized platform for maintain and develop the codes and track the team member's contributions. The tasks are distributed equally and all of the team members are contributed to the repository. First of all the basic operations such as web cam and landmark configurations are developed together as a team and after that GUI, visual feedbacks are developed. In all stages of developing, team handled bugs. The general workflow was as following Erdem primarily created the instruction page, laying the foundation for clear guidance within the project and also integrated the landmark functions to the game. Adnan handled the initial setup, establishing essential project configurations. The development of the hand-based application and webcam integration was a collaborative effort involving Zeki, Adnan, Erdem, and İlyas, who worked together to create a reliable and functional system. Zeki took the lead in implementing the quit interaction feature by using three hand which exit process. The homepage and play game features were designed by Adnan, contributing to the project's user experience. Visual feedback elements, such as buttons changing to a blue color upon being clicked, were crafted by Zeki and İlyas, enhancing the interface's interactivity. The project was finalized collectively, with all team members contributing to polishing and completing the code.

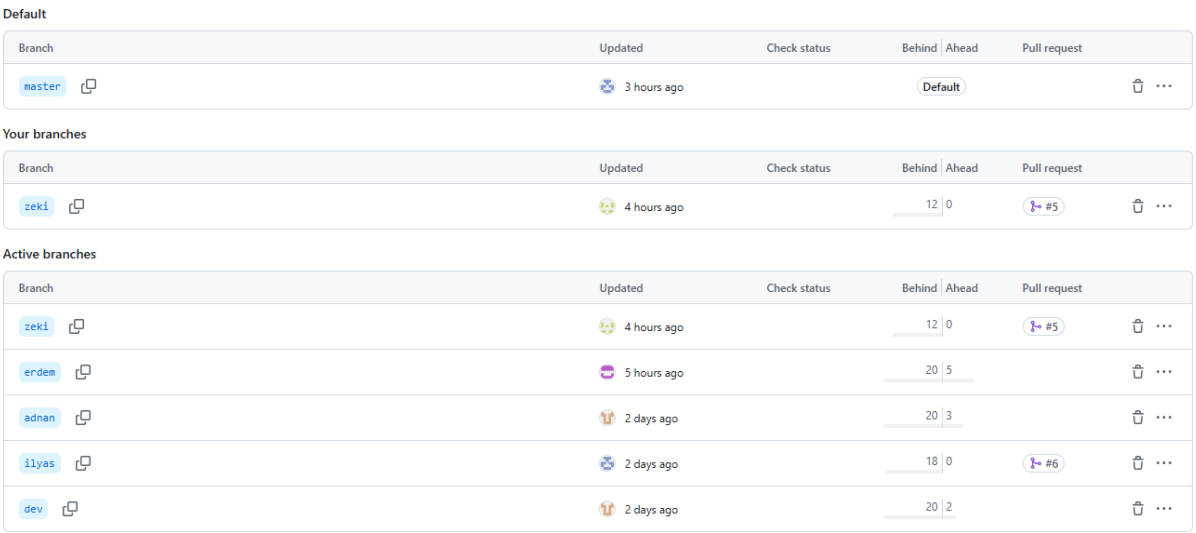

Figure 11 : Github Branch Names

As can be seen in the figure all team members developed the application by using different branches and after each member obtain a valid program member commited the changes and pushed to the master branch. By this systematic approach team completed the project effectively.

5. Video Demo

Youtube Link: https://www.youtube.com/watch?v=CKM27KB-BMo

6. References:

- Landauer, T. K. (1995). The trouble with computers: Usefulness, usability, and productivity. MIT Press.
- Norman, D. A. (2013). The design of everyday things (Revised and expanded edition). Basic Books.
- Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., & Elmqvist, N. (2016). Designing the user interface: Strategies for effective human-computer interaction (6th ed.). Pearson.

7. Contributors

Zeki Karamuk 22496

Erdem Özcan 29225

Mustafa Adnan Arasan 29413

İlyas yeşilyaprak 29294